02P17494

2006Q07474

8286 Computers in Industry
24(1994)May, No.1, Amsterdam, NL

B4

G06F15/40

XP 000454379

ELSEVIER

Computers in Industry 24 (1994) 17–28 (12)

COMPUTERS IN INDUSTRY

Product modelling using multiple levels of abstraction Instances as types

Frederik Erens [a,b,*], Alison McKay [c], Susan Bloor [c]

[a] *Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*
[b] *KPMG Lighthouse, P.O. Box 6427, 5600 HK Eindhoven, The Netherlands*
[c] *Department of Mechanical Engineering, University of Leeds, Leeds LS2 9JT, UK*

(Received August 10, 1993; accepted November 8, 1993)

Abstract

Most databases make use of three levels of abstraction, namely: the data dictionary, the database schema and the database contents. The data dictionary describes the structure of the database schema whilst the database schema describes the structure of the database contents. This approach fits perfectly in situations with large quantities of "simple" data and relatively small and stable structures. In this paper, we will focus on "product models" which cannot be modelled easily with these levels of abstraction. We will illustrate the modelling problem with an example and present a solution using the Leeds Product Data Editor.

1. Problem statement

The use of databases is possibly as old as writing. Both merchants and armies created lists of formatted data, more than 2000 years ago. Once the list (database) structure was defined, data could be added to the list by filling in the necessary attributes. This century the use of formatted data has increased dramatically, especially in governmental organizations and large companies. Information about citizens, for example, is stored in tens of databases with hundreds of attributes describing specific characteristics of these citizens. Company-owned databases for customers, suppliers and employees add even more attributes to this list. All these databases have a relatively stable database schema in that the structure of the data that is stored will not change considerably over time. This stability is accompanied by large quantities of repetitive data, often thousands or millions of occurrences of a certain structure. Computer-based information systems that are built on top of these databases are especially good in processing these large quantities of data.

The last decade, however, has shown a considerable growth of nontraditional applications, like personal information management systems, computer-aided co-operative work systems and computer-aided engineering systems [1]. Features that

* Corresponding author.

8286 Computers in Industry
24(1994)May, No.1, Amsterdam, NL

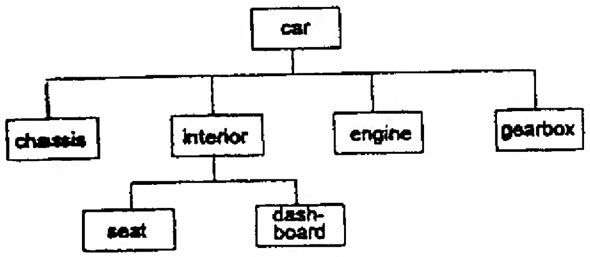

Fig. 2. Product family structure.

specific (manufacturing) documentation for that specification (Fig. 1) [2].

In order to enable the generation of a product variant description, specifically for a given customer specification, a product family description must be available. The creation of such a family description is normally the responsibility of marketing, engineering and design. It is an example of data where structural data is intertwined with repetitive data and shared between different users.

Example

Consider, for example, a car family. A car is, amongst other things, assembled from a chassis, an engine, a gearbox and an interior. The interior is assembled from the seats and the dashboard (see Fig. 2). The relationships between the different parts of the car are structural data, while the parts themselves are repetitive data which may itself be structured.

Further, a number of views can be determined, each having its specific requirements in the organization. The design process of such a car involves two different types of activity and produces data on different abstraction levels, namely:
- "structural data", shared by all cars of that family which defines the structure of a car;
- "repetitive data", which details the components of a car.

These different abstraction levels will be elucidated in the next section. First we will continue with this example.

Each of the aforementioned sub-assemblies can be considered as a product family. The interior, for example, is a product family which is assembled from the seats and the dashboard. All product families have a number of variants. The variety of the car family originates from the variety of its sub-assemblies. In turn, the variety of the interior originates from its sub-assemblies, i.e. of the seats and the dashboard.

The data structures behind today's engineering databases [3,4] typically necessitate the definition of every variant of a product family. This means the need for the separate description of possibly millions of variants of a car, where each description has a large commonality with descriptions of other variants. This redundancy of data is normally not acceptable.

When variants of a product family are physically assembled from component variants, it is possible to build the descriptions of the variants. The description of a car variant can be built from:
- the descriptions of the chassis, engine, gearbox and interior variants; plus
- the description of the car family (e.g. how do the component families fit together).

For example, the description of an interior variant can be built from the descriptions of the seat and dashboard variants, plus some information about interiors in general (e.g. how seat and dashboard families are normally assembled). Fig. 3 shows the built descriptions in black and the basic descriptions in white. Basic descriptions belong to basic, or so-called primary variants, while built descriptions belong to so-called assembly variants.

For the purpose of this paper, we will assume that all assemblies are assembled to customer order. The generation of product descriptions for

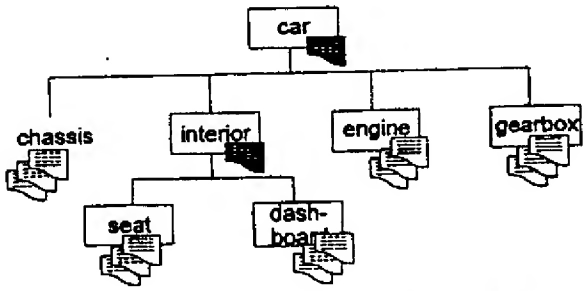

Fig. 3. Building documentation of assembled variants.

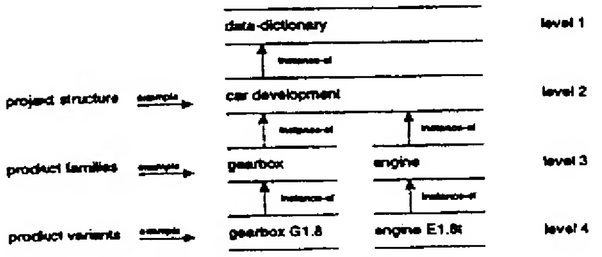

Fig. 4. Examples of classification.

Limitations of conventional database systems

Redundancy of data is introduced when these abstraction levels are implemented in separate three-layer databases:

The first type of redundancy concerns the different abstraction levels. It is not possible to model four levels of abstraction without repeating structural data in one database as repetitive data in another database. Conventional relational or object-oriented databases have only three levels, namely: data dictionary, tables/classes and tuples/instances.

The second type of redundancy concerns redundancy within a single level. Database tables (for example for variants of the engine and the gearbox) will have a number of similar attributes, which are repeated for tables. An example is given in Section 4.1.

The next section describes how today's techniques deal with the modelling of abstraction levels and data redundancy in general. It is shown that even modern object-oriented databases are not powerful enough to model this problem conceptually right. More advanced, but still academic, approaches offer better solutions.

4. Current practices and insights

In this section, we will describe three different approaches before we elaborate a fourth approach, which is based on the Leeds Product Data Editor, in Section 5. Firstly, we will discuss an implementation in a relational database. Secondly, we will describe the advantages and shortcomings of object-oriented databases. Finally, we will show a fundamentally different approach (3DIS) which has been proposed by Afsarmanesh and McLeod [1].

4.1. Relational approach

Relational databases make a clear distinction between data and structure [6]. Normally, there is a large volume of data and only a relatively small and stable structure. This structure is often called the database schema and is used by application programmes to work on the data. According to our previous figure, we will need a number of different, however related databases:

- "car development" as a project database for "specific product families" (see Table 2);
- several databases for product families, e.g. "engine" and "gearbox" (see Tables 3 and 4).

From this example, it can be seen that "engine" and "gearbox" appear both on the data level (Table 2), and on the schema level (Tables 3 and 4).

In this way data redundancy over different levels of abstraction is introduced. Further, the engine (gearbox) information that is presented in Table 2 is valid for all design variants of that engine (gearbox) which can be found in Table 3

Table 2
Attributes and data for "car development"

Product family name	Responsibility	Family start date	Family finish date	Main product relationships
Car	P. Breuls	1992-10	1995-2	Engine, Gearbox
Engine	H. Hegge	1993-3	1994-8	Ignition, Gearbox
Gearbox	E. Platier	1993-3	1994-6	Interior, Engine
Interior	R. Stekel	1993-7	1994-11	Dashboard, Seats
Chassis	T. Renkema	1992-10	1993-12	Interior, Suspension

8286 Computers in Industry
24(1994)May, No.1, Amsterdam, NL

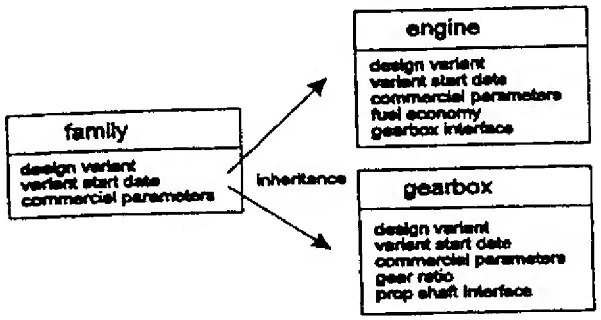

Fig. 5. Inheritance of structural information.

4.3. 3DIS approach

Afsarmanesh and McLeod [1] propose a database where all information including the data, the descriptions and classifications of data (meta-data), abstractions, operations, and constraints are treated uniformly as objects. Within this framework, 3DIS incorporates several predefined abstraction primitives, including a generalization hierarchy.

Objects and mappings are the two basic modelling constructs in 3DIS. Relationships among objects are modelled by "(domain-object, mapping-object, range-object)" triples. The structure of these triples is very much like the binary relationships in the binary semantic model [8]. Below some mappings are given, which are used for the car example in Figs. 6–8:

Member/type mappings
- instantiation: Has-member: type objects → P(atomic/composite objects)
- classification: (Is-a-member-of)

Member-mapping/type mappings
- decomposition: Has-member-mapping: type objects → P(member-mappings)
- aggregation: (Is-a-member-mapping-of)

```
( Product, Has-member-mapping, {
                              Has-product-family-name,
                              Has-responsibility,
                              Has-family-start-date,
                              Has-family-finish-date,
                              Has-main-product-relationships } )
```

Fig. 6. Car development level.

```
( Engine, Has-member-mapping, {   Has-design-variant,
                                  Has-fuel-economy,
                                  Has-gearbox-interface,
                                  Has-variant-start-date,
                                  Has-commercial-parameters } )

( Engine, Is-a-member-of, { Product } )

( Commercial-parameters, Has-member-mapping, {  Has-name,
                                                Has-value } )

( Engine, Has-related-product, { Ignition } )

( Engine, Has-related-product, { Gearbox } )

( Engine, Has-member, {  E1.8,
                         E2.0,
                         E1.8t,
                         E2.0t } )
```

Fig. 7. Engine definition level.

Every identifiable information fact in an application environment corresponds to an object in a 3DIS database. Simple, compound, and behavioural entities in an application environment, attributes of objects and relationships among objects, as well as object groupings and classifications are all modelled as objects. What distinguishes different kind of objects in a 3DIS database is the set of structural and nonstructural (data) relationships defined on them.

The concept which we have exploited to model multiple levels of abstraction is "instantiation/classification". However the semantics of these terms are not defined in Ref. [1]. Our interpretation is that an instance of a class is an object whose type is the class and which has values for the attributes of that class. For example, *Engine (1.8 litre, with turbo)* is an instance of the object engine defined as *Engine HAS (size (1.8 litre OR 2.0 litre) AND turbo (with OR without))*.

```
( E1.8, Is-a-member-of, { Engine } )

( E1.8, Has-fuel-economy, { 16 km/litre } )

( E1.8, Has-gearbox-interface, { G1.8 } )

( E1.8, Has-variant-start-date, { 1993-3 } )

( E1.8, Has-commercial-parameters, { Engine-size=1.8, Turbo=no } )
```

Fig. 8. Engine instance level (E1.8 only).

8286 Computers in Industry
24(1994)May, No.1, Amsterdam, NL

5.2. Using λ-calculus for modelling multiple abstraction levels

The solution adopted in the structure editor is to model the form of final instances, i.e. the lowest abstraction level, in the meta-structure, and to model the intermediate abstraction levels with λ-calculus. This means for our purposes that:
- the structure and content of a product variant are modelled in the meta-structure (level 2 in Fig. 4);
- specific variants for customer orders are modelled as instances of the meta-structure (level 4 in Fig. 4);
- product families as cars, trucks and aeroplanes are modelled using λ-calculus (level 3 in Fig. 4).

Fig. 10 shows the meta-structure for variants. This meta-structure is defined recursively because components of a variant can be variants again. The LIST construct is used to model an arbitrary number of components, which are all variants again.

Fig. 11 shows a small part of a car assembly variant (coded AV11) and its engine (coded E1.8t). This variant is an instance of the meta-structure of Fig. 10. Now, we will use this specific instance to model the car family with its component families. We will replace specific instances by a parametrized λ-function. The parameters of this function are used to define the precise primary variants of a family. An example will clarify this.

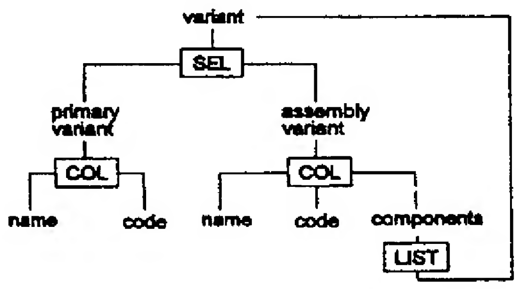

Fig. 10. Meta-structure for variants.

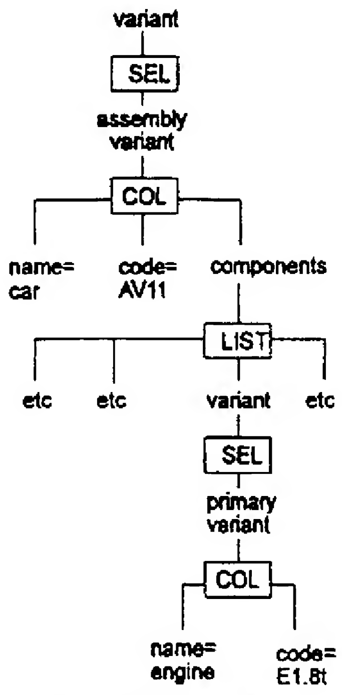

Fig. 11. A specific car variant.

First we will look at a component family of the car, namely the engine. The engine has four variants, of which the selection is dependent on the commercial parameters "engine size" and "turbo". If we model the engine as a function, the abovementioned commercial parameters will be used to invoke this function.

```
Engine( engine size = [1.8, 2.0],
        turbo = [yes, no])
```

with body:

```
IF engine size = 1.8 AND turbo = no
  THEN E1.8
IF engine size = 2.0 AND turbo = no
  THEN E2.0
IF engine size = 1.8 AND turbo = yes
  THEN E1.8t
IF engine size = 2.0 AND turbo = yes
  THEN E2.0t
```

interface supports this interaction. 3DIS, on the other hand, is a database system which is best used in conjunction with sophisticated, and possibly data structure specific, user and application interfaces. In practice both systems have a role to play: the SE allows people to prototype and test their data structures whereas 3DIS provides a database environment in which the tested data structures can be implemented and used in a production environment.

6. Conclusions

This paper has tried to raise an awareness with respect to abstraction levels. The development of products, and product families in particular, shows that there are modelling problems which cannot be solved directly in relational or object-oriented databases. Forcing multiple abstraction levels in three database levels introduces more complex schema with inherent redundancy. Although this problem is generally known with experienced database designers, not much attention has been paid to this subject in literature or software design. Two academic approaches, the 3DIS and the Leeds Product Data Editor, have been developed to tackle multiple abstraction levels. They appear to be appropriate for modelling product families, but supported software of this kind is not yet available.

Acknowledgement

This work has been partly funded by the ACME Directorate of SERC Grant No. GR/H 24266.

References

[1] H. Afsarmanesh and D. McLeod, "The 3DIS: An extensible object-oriented information management environment", *ACM Trans. Inf. Syst.*, Vol. 7, No. 4, October 1989, pp. 339–377.
[2] E.A. van Veen, *Modelling Product Structures by Generic Bills-of-Materials*, Elsevier, Amsterdam, 1992.
[3] STEP Part 44, ISO-10303-44, Product Data Representation and Exchange, Integrated Generic Resources: Product Structure Configuration, Draft International Standard, 1993.
[4] P. Gu, " PML: Product modelling language", *Computers in Industry*, Vol. 18, 1992, pp. 265–277.
[5] F.J. Erens, H.M.H. Hegge, E.A. van Veen and J.C. Wortmann, "Generative bills-of-material: An overview", *Proc. IFIP'92*, Elsevier, Amsterdam, 1992.
[6] J.J. van Griethuysen, "Concepts and terminology for the conceptual schema and the information base", ISO/TC97/SC5 N695, 1982.
[7] W. Kim (Ed.), *Object-Oriented Concepts, Databases and Applications*, ACM Press, 1989.
[8] J.R. Abrial, "Data semantics", in J.W. Klimbie and K.L. Koffeman (Eds.), *Data Management Systems*, North-Holland, Amsterdam, 1974.
[9] C. Batini, M. Lenzerini and S.B. Navathe, "A comparative analysis of methodologies for database schema integration", *ACM Comput. Surv.*, Vol. 18, No. 4, December 1986, pp. 323–364.
[10] A. McKay, "The Structure Editor approach to product description", University of Leeds, ISS Project Research Report, ISS-PDS-Report-4, June 1988.

Frederik Erens gained his degree in Information Science in 1990, after which he became a consultant at KPMG Lighthouse. Simultaneously, he started as a Research Engineer in the Department of Industrial Engineering and Management Sciences, Eindhoven University of Technology. His current research interests include product modelling, integral logistics and the development of complex products which are offered in a large variety.

Alison McKay gained her degree in Mechanical Engineering in 1982 and then spent two years in industry before becoming a Research Engineer in the Department of Mechanical Engineering. Her research interests include the representation and integration of product data. Since 1988 she has taken an active role in STEP, particularly in integration, development methods and as a editor of Part 41. Currently she is leading the product data modelling and integration activities of the SERC/ACME-funded MOSES (Model Oriented Simultaneous Engineering Systems) project.